JOURNAL OF COMPLEXITY **18**, 912–942 (2002)
doi:10.1006/jcom.2002.0648

Topological Complexity of Zero Finding with Algebraic Operations

Peter Hertling[1]

Theoretische Informatik I, Fernuniversität Hagen, 58084 Hagen, Germany
E-mail: peterhertling@fernuni-hagen.de

Received June 30, 2000; revised April 3, 2002; accepted April 4, 2002;
published online July 30, 2002

How many tests does one have to perform in order to compute an ε-approximation of a zero of a function f out of a given class of continuous functions on the unit interval, for a given $\varepsilon > 0$? We study this question in the context of the real number oracle machine model which uses the four standard arithmetic operations, comparisons with zero, and function values at adaptively chosen points. Let $0 < \varepsilon < \frac{1}{2}$. We show that for the class of all continuous functions f on the unit interval with $f(0) < 0$ and $f(1) > 0$ one needs exactly $\lceil \log_2(1/(2\varepsilon)) \rceil$ tests in the worst case during a computation. For the subclass of all functions which are additionally nondecreasing one needs roughly $\log_2 \log_2 \varepsilon^{-1}$ tests, and for the subclass of all functions which are additionally increasing one needs exactly 1 test. © 2002 Elsevier Science (USA)

Key Words: topological complexity; zero finding; algebraic operations; oracle machine over the real numbers; information-based complexity.

1. INTRODUCTION

The notion "topological complexity" was introduced by Smale [6], who gave estimations for the total number of comparison nodes in a computation tree that computes a vector of zeros of a univariate complex polynomial up to a small error. This and similar problems and their relations to algebraic topology have been investigated further by Vassiliev and others; for an overview see [7]. Novak and Woźniakowski [5] and the author [3] studied the topological complexity of the problem to approximate a zero of a function out of a given class of continuous functions f on the unit interval with $f(0) < 0$ and $f(1) > 0$ with a prescribed precision ε. Here we continue the work done in [5, 3]. In [5,3] both (a) the total number of comparison nodes in a computation tree and (b) the worst case number of

[1] Part of the work on this paper was done while the author was supported by a DFG Research Scholarship and a guest at the MSRI, Berkeley, California, and part was done while the author was supported by Grant 221-97-745 of the Swedish Research Council for Engineering Sciences and a guest at the Department of Mathematics at Uppsala University, Sweden.

0885-064X/02 $35.00

comparisons performed during a computation, i.e., the number of comparisons on the longest computation path in a computation tree, were studied. Computation trees derived from real number oracle machines were considered which could (1) use arithmetic operations out of a given class of arithmetic operations, (2) use arbitrary real constants, (3) compare real numbers with 0, and (4) ask for the function values of the input function f at adaptively chosen points. The following classes of arithmetic operations were considered: the class $\{+,-,*,/,|\cdot|\}$ where $|\cdot|$ stands for the absolute value function, the class $\{+,-,*,/,\log,\exp\}$, the class of all continuous operations, and the class of all operations which satisfy a Hölder condition on each bounded subset of their domain. But the question how many comparison nodes are needed if only the standard arithmetic operations $\{+,-,*,/\}$ are allowed, was still open. In this paper, we answer this question for three classes of continuous functions f on the unit interval with $f(0)f(1)<0$. In the following, we assume $0<\varepsilon<\frac{1}{2}$.

We show that for the class of all continuous functions f on the unit interval with $f(0)f(1)<0$ there is a balanced computation tree using $\{+,-,*,/\}$ which contains exactly $\lceil 1/(2\varepsilon)\rceil-1$ comparison nodes in total and exactly $\lceil \log_2(1/(2\varepsilon))\rceil$ comparison nodes on the longest computation path. On the other hand, we show that this is optimal: any computation tree using $\{+,-,*,/\}$ needs $\lceil 1/(2\varepsilon)\rceil-1$ comparison nodes in total and therefore at least $\lceil \log_2(1/(2\varepsilon))\rceil$ comparison nodes on the longest computation path. This is exponentially worse than for the class $\{+,-,*,/,|\cdot|\}$ where one needs roughly $\log_2 \varepsilon^{-1}$ comparison nodes in total, and roughly $\log_2 \log_2 \varepsilon^{-1}$ comparisons on the longest computation path [3]. The proof of the lower bound borrows an idea used by Vassiliev [8] and by Grigoriev *et al.* [2] related to the Newton polyhedron of a polynomial.

For the subclass of nondecreasing functions the author [3] had shown that any computation tree using the arithmetic operations $\{+,-,*,/,|\cdot|\}$ has at least roughly $\log_2 \varepsilon^{-1}$ comparison nodes in total, and therefore at least roughly $\log_2 \log_2 \varepsilon^{-1}$ comparisons on the longest computation path, and that on the other hand there is a balanced computation tree with exactly these numbers of comparisons (in total and on the longest computation path). Here we show that restricting oneself to $\{+,-,*,/\}$ instead of $\{+,-,*,/,|\cdot|\}$ leads only to an insignificant change: one needs at most one more comparison in total or on the longest computation path.

Finally, we show that for the subclass of increasing functions one needs exactly one comparison if one allows the operations $\{+,-,*,/\}$. This is in contrast to the fact that for the subclass of increasing functions one can solve the problem without any comparison if instead of $\{+,-,*,/\}$ the larger class of operations $\{+,-,*,/,\log,\exp\}$ is allowed [5], or the class of operations $\{+,-,*,/,|\cdot|\}$ is allowed [3].

In the following section, we present the results in detail, after giving a short, slightly more detailed description of the computation model and after defining formally the considered problems and their topological complexity. In Section 3, we prove the upper bounds for the complexities stated in Section 2, and in Section 4 we prove the lower bounds.

2. THE RESULTS

2.1. The Computation Model

Before we describe the problem and present the results in detail we give a short description of the computation model.

The input for our algorithms are functions mapping real numbers to real numbers. As the computation model we use the real number oracle machine model as described in detail by Novak [4] and by Novak and Woźniakowski [5]. It is close to the real number machine model of Blum *et al.* [1]. Such a real number oracle machine can be described by a flowchart which can be developed into a computation tree. Each input leads to a computation path starting at the root and ending in one of the leaves. In the leaves the result is produced. The internal nodes may be unary or binary. In the unary nodes either a real constant is introduced, or a new value is computed by applying an arithmetic operation to already computed values, or the input function f is evaluated at some previously computed value v. Note that this value v may depend on previously obtained values of f. Hence, we allow adaptive function evaluation. In the binary nodes a previously computed value v is compared with zero: "$v \geqslant 0$?". We also allow tests of the form "v rel 0?" with rel $\in \{\leqslant, >, <\}$. But it is clear that they can be expressed via tests of the first form.

In this paper, as arithmetic operations we will allow only the standard arithmetic operations addition, subtraction, multiplication, and division, written shortly as $\{+, -, *, /\}$. Since we are interested in finding the minimum number of comparisons needed by any algorithm for a specific problem, we are interested only in algorithms with a finite computation tree. In order to stress that we consider only the operations $\{+, -, *, /\}$ and consider only finite computation trees, we will usually speak about *algebraic oracle computation trees.* Note that all internal nodes of these trees are unary or binary. Trees with this property will also be called *binary* trees. Any binary tree with n binary nodes in total has at least one path (from the root to a leaf) which contains at least $\lceil \log_2 (n+1) \rceil$ binary nodes. If a binary tree has n binary nodes in total and any path in the tree contains at most $\lceil \log_2 (n+1) \rceil$ binary nodes, then we call the tree *balanced.*

2.2. The Problem

We come to the precise description of the problem. A function $f:[0,1]\to\mathbb{R}$ is called *nondecreasing*, if for $x,y\in[0,1]$, $x<y$ implies $f(x)\leqslant f(y)$. It is called *increasing*, if for $x,y\in[0,1]$, $x<y$ implies $f(x)<f(y)$. We consider the following three classes of functions:

$$F=\{f\in C[0,1]\mid f(0)\cdot f(1)<0\},$$

$$F_{\mathrm{nd}}=\{f\in F\mid f \text{ is nondecreasing}\},$$

$$F_{\mathrm{inc}}=\{f\in F\mid f \text{ is increasing}\}.$$

Obviously, $F_{\mathrm{inc}}\subsetneq F_{\mathrm{nd}}\subsetneq F$. We also consider the following subclasses:

$$G^{-1,1}=G\cap\{f\in C[0,1]\mid f(0)=-1 \text{ and } f(1)=1\}$$

for $G\in\{F,F_{\mathrm{nd}},F_{\mathrm{inc}}\}$. A real number $x\in[0,1]$ is a *zero* of a function $f:[0,1]\to\mathbb{R}$ if $f(x)=0$. For $\varepsilon>0$, an *ε-approximation* of a number x^* is a number x with $|x-x^*|\leqslant\varepsilon$. We wish to approximate zeros of functions $f\in F$ in the *root error* sense [5]. That means, for some $\varepsilon>0$, and given a function $f\in F$, we wish to compute a real number which is an ε-approximation of some zero of f.

The computational problems we consider are defined by fixing some precision $\varepsilon>0$ and a function class G with $F_{\mathrm{inc}}^{-1,1}\subseteq G\subseteq F$. We consider algebraic oracle computation trees which, on input $f\in G$, compute an ε-approximation of some zero of f. We ask for the minimal number of comparisons which an algebraic oracle computation tree has to perform to solve this problem. We will consider two different kinds of "minimal number of comparisons." The term *topological complexity* is used for both of them. One can either count the total number of comparison nodes in the computation tree:

$$\mathrm{comp}_{\mathrm{total}}(G,\varepsilon):=\min_{\text{trees } A}\ \#\ (\text{comparison nodes in } A),$$

where the minimum is taken over all algebraic oracle computation trees that compute an ε-approximation of a zero of f for all f from G. Or one can consider the *depth* of the computation tree, which we define to be the largest number of comparison nodes on any path in the computation tree:

$$\mathrm{comp}_{\mathrm{path}}(G,\varepsilon):=\min_{\text{trees } A}\ \max_{\text{paths } p \text{ in } A}\ \#\ (\text{comparison nodes on } p),$$

where again the minimum is taken over all algebraic oracle computation trees that compute an ε-approximation of a zero of f for all f from G. Note

that a lower bound for the total number of comparison nodes in any computation tree for the problem implies a logarithmic lower bound for the depth of any computation tree for the problem since our computation trees are binary; cf. Section 2.1:

$$\lceil \log_2(\mathrm{comp}_{\mathrm{total}}(G,\varepsilon)+1)\rceil \leqslant \mathrm{comp}_{\mathrm{path}}(G,\varepsilon).$$

For certain function classes G, we shall give (almost) sharp lower bounds for the total number $\mathrm{comp}_{\mathrm{total}}(G,\varepsilon)$ of comparison nodes in any computation tree and show that the resulting logarithmic lower bounds for the depth $\mathrm{comp}_{\mathrm{path}}(G,\varepsilon)$ of any computation tree are (almost) sharp as well since there exist optimal balanced algorithms.

2.3. *The Results*

The following theorem contains the main results of this paper.

THEOREM 1. *Let a number $\varepsilon > 0$ be fixed.*

(1) *For $F^{-1,1} \subseteq G \subseteq F$ we have*

$$\mathrm{comp}_{\mathrm{total}}(G,\varepsilon) = \lceil 1/(2\varepsilon)\rceil - 1,$$

$$\mathrm{comp}_{\mathrm{path}}(G,\varepsilon) = \lceil \log_2(\lceil 1/(2\varepsilon)\rceil)\rceil,$$

(2) *For $F_{\mathrm{nd}}^{-1,1} \subseteq G \subseteq F_{\mathrm{nd}}$ we have*

$$\lceil \log_2(\varepsilon^{-1}+2)\rceil - 2 \leqslant \mathrm{comp}_{\mathrm{total}}(G,\varepsilon) \leqslant \lfloor \log_2(1/\min\{1,\varepsilon\})\rfloor,$$

$$\lceil \log_2(\lceil \log_2(\varepsilon^{-1}+2)\rceil - 1)\rceil \leqslant \mathrm{comp}_{\mathrm{path}}(G,\varepsilon) \leqslant \lceil \log_2(\lfloor \log_2(1/\min\{1,\varepsilon\})\rfloor + 1)\rceil.$$

(3) *For $F_{\mathrm{inc}}^{-1,1} \subseteq G \subseteq F_{\mathrm{inc}}$ we have*

$$\mathrm{comp}_{\mathrm{total}}(G,\varepsilon) = \mathrm{comp}_{\mathrm{path}}(G,\varepsilon) = \begin{cases} 0 & \text{if } \varepsilon \geqslant \frac{1}{2}, \\ 1 & \text{if } \varepsilon < \frac{1}{2}. \end{cases}$$

The lower bounds in the second part of the theorem are already essentially known and have been added only for completeness: the lower bound

$$\lceil \log_2(\varepsilon^{-1}+2)\rceil - 2 \leqslant \mathrm{comp}_{\mathrm{total}}(F_{\mathrm{nd}},\varepsilon)$$

was already shown by the author [3], and the proof given there applies also to the subclass $F_{\mathrm{nd}}^{-1,1}$. The proofs of the other results will be given in the following sections.

As we remarked above, for each lower bound b for $\mathrm{comp}_{\mathrm{total}}(G,\varepsilon)$ the number $\lceil \log_2(b+1)\rceil$ is a lower bound for $\mathrm{comp}_{\mathrm{path}}(G,\varepsilon)$. On the other hand, as one reads off the theorem, for each of the upper bounds b for $\mathrm{comp}_{\mathrm{total}}(G,\varepsilon)$ given in the theorem, the number $\lceil \log_2(b+1)\rceil$ is an upper bound for $\mathrm{comp}_{\mathrm{path}}(G,\varepsilon)$. Furthermore, we will see that in each case the upper bounds are proved by balanced algorithms, hence, by computation trees which prove both the upper bound b for $\mathrm{comp}_{\mathrm{total}}(G,\varepsilon)$ and the upper bound $\lceil \log_2(b+1)\rceil$ for $\mathrm{comp}_{\mathrm{path}}(G,\varepsilon)$ at the same time.

Because of

$$\lceil \log_2(\varepsilon^{-1}+2)\rceil - 2 \leqslant \lfloor \log_2(1/\min\{1,\varepsilon\})\rfloor \leqslant \lceil \log_2(\varepsilon^{-1}+2)\rceil - 1, \tag{1}$$

the lower and upper bounds in the second part of the theorem for $F_{\mathrm{nd}}^{-1,1} \subseteq G \subseteq F_{\mathrm{nd}}$ differ at most by 1.

It is interesting to note that the upper bounds for the class F are achieved by an algorithm which, in case $\lceil 1/(2\varepsilon)\rceil = 2^n$ for some n, is just the bisection algorithm. Thus, for such ε the bisection algorithm is optimal with respect to the total number of comparisons in the tree (among all algorithms which use only the four basic arithmetic operations $\{+,-,*,/\}$ and compute an ε-approximation of a zero for input functions $f \in F$). And for arbitrary $\varepsilon > 0$, the bisection algorithm with $\lceil \log_2(\lceil 1/(2\varepsilon)\rceil)\rceil$ comparison nodes on each path is optimal (among all such algorithms) with respect to the depth of the tree, i.e., with respect to the maximal number of comparisons during a computation.

2.4. Comparison of the Results

It is interesting to compare the results above with the known results for other classes of arithmetic operations. For a set ARI of real-valued functions on real numbers, let us define

$$\mathrm{comp}_{\mathrm{total}}(G,\mathrm{ARI},\varepsilon) \quad \text{and} \quad \mathrm{comp}_{\mathrm{path}}(G,\mathrm{ARI},\varepsilon)$$

as above, but with respect to oracle computation trees which, instead of $\{+,-,*,/\}$, may use arithmetic operations from the set ARI. Table I contains the results for $\mathrm{comp}_{\mathrm{total}}(G,\mathrm{ARI},\varepsilon)$ for classes ARI with

$$\{+,-,*,/,|\cdot|\} \subseteq \mathrm{ARI} \subseteq \{\text{arbitrary continuous operations}\}$$

(obtained by the author [3]) and for the class $\mathrm{ARI} = \{+,-,*,/\}$.

Furthermore, for the class F_{inc}, Novak and Woźniakowski [5] have shown

TABLE I
Results for $\mathrm{comp}_{\mathrm{total}}(G, \mathrm{ARI}, \varepsilon)$

	$\{+,-,*,/,\lvert\cdot\rvert\} \subseteq \mathrm{ARI} \subseteq \{\text{arb. cont. op.}\}$	$\mathrm{ARI} = \{+,-,*,/\}$
$G = F_{\mathrm{inc}}$	0	1
$G = F_{\mathrm{nd}}$	$\lceil \log_2(\varepsilon^{-1}+2)\rceil - 2$	$\begin{cases}\text{lower bound}: \lceil \log_2(\varepsilon^{-1}+2)\rceil - 2 \\ \text{upper bound}: \lfloor \log_2(1/\min\{1,\varepsilon\})\rfloor\end{cases}$
$G = F$	$\lceil \log_2(\varepsilon^{-1}+2)\rceil - 2$	$\lceil 1/(2\varepsilon)\rceil - 1$

$$\mathrm{comp}_{\mathrm{total}}(F_{\mathrm{inc}}, \{+,-,*,/,\exp,\log\}, \varepsilon) = 0.$$

Thus, not allowing to use exp and log or the absolute value function $x \to |x|$ in addition to the basic four arithmetic operations $\{+,-,*,/\}$ makes it necessary to use a comparison for the class F_{inc}. This is a small quantitative difference, but an interesting qualitative difference. For any function class G with $F_{\mathrm{nd}}^{-1,1} \subseteq G \subseteq F_{\mathrm{nd}}$ we see that due to Eq. (1) it makes only a small quantitative difference (at most one comparison) whether one may use only the four basic arithmetic operations or additionally also the absolute value function or even arbitrary continuous operations. But for the (largest) classes G with $F^{-1,1} \subseteq G \subseteq F$ the difference between $\mathrm{comp}_{\mathrm{total}}(G, \{+,-,*,/\}, \varepsilon)$ and $\mathrm{comp}_{\mathrm{total}}(G, \{+,-,*,/,|\cdot|\}, \varepsilon)$ is exponential.

One obtains a table for $\mathrm{comp}_{\mathrm{path}}(G, \mathrm{ARI}, \varepsilon)$ corresponding to Table I by replacing each value b in Table I by $\lceil \log_2(b+1)\rceil$. Therefore, all statements in this subsection (Section 2.4) made with respect to $\mathrm{comp}_{\mathrm{total}}(G, \mathrm{ARI}, \varepsilon)$ are also true for $\mathrm{comp}_{\mathrm{path}}(G, \mathrm{ARI}, \varepsilon)$.

2.5. *Possible Future Work*

We conclude this section with some remarks on possible further work.

For the class F_{nd} of nondecreasing functions we simply cited the lower bound from [3] and did not try to determine the precise complexities since the difference between the lower and the upper bound is at most one, hence it is insignificant. Nevertheless, one may try to determine the exact complexities $\mathrm{comp}_{\mathrm{total}}(F_{\mathrm{nd}}, \varepsilon)$ and $\mathrm{comp}_{\mathrm{path}}(F_{\mathrm{nd}}, \varepsilon)$.

In this paper we have analyzed only the required number of comparisons. One might also analyze it in connection with other measures for the complexity, like the required number of function evaluations or arithmetic operations, and investigate whether there is a tradeoff. The algorithms which we present in Section 3 for the class of increasing functions (using only one

test) and for nondecreasing functions (using only roughly $\log_2 \log_2 \varepsilon^{-1}$ tests during the longest computation) use in the worst case $\Theta(\varepsilon^{-1})$ function evaluations during a computation. That is exponentially worse than bisection.

Finally, one might analyze the problems above also for other classes of arithmetic operations, e.g., the class $\{+, -, *, /, \exp, \log\}$ considered so far only in combination with F_{inc} [5].

3. THE UPPER BOUNDS

In this section we prove the upper bounds for the complexities which were stated in Theorem 1.

For $\varepsilon \geqslant \frac{1}{2}$ the problem to find an ε-approximation of a zero of a function f in a class G of functions is trivial even for the class $G = F$: the algorithm which simply always gives the number $\frac{1}{2}$ as the result certainly computes an ε-approximation of a zero for any function $f \in F$. Hence, for $G \subseteq F$ and $\varepsilon \geqslant \frac{1}{2}$,

$$\text{comp}_{\text{total}}(G, \varepsilon) = \text{comp}_{\text{path}}(G, \varepsilon) = 0.$$

One checks that this proves the three assertions of Theorem 1 in the case $\varepsilon \geqslant \frac{1}{2}$. In the following subsections we consider the case $0 < \varepsilon < \frac{1}{2}$ and give algorithms which prove the assertions in this case. For the upper bounds it is sufficient to consider the classes F, F_{nd} and F_{inc}.

3.1. The Algorithm for Arbitrary Functions

In this section, we prove the upper bounds

$$\text{comp}_{\text{total}}(F, \varepsilon) \leqslant \lceil 1/(2\varepsilon) \rceil - 1$$

and

$$\text{comp}_{\text{path}}(F, \varepsilon) \leqslant \lceil \log_2(\lceil 1/(2\varepsilon) \rceil) \rceil$$

for arbitrary ε with $0 < \varepsilon < \frac{1}{2}$. We show that for arbitrary $0 < \varepsilon < \frac{1}{2}$ there is an algebraic oracle computation tree that has only $\lceil 1/(2\varepsilon) \rceil - 1$ branching nodes in total and at most $\lceil \log_2(\lceil 1/(2\varepsilon) \rceil) \rceil$ branching nodes on any path and computes an ε-approximation of some zero of f, for any $f \in F$.

We shall describe the algorithm as a binary tree and use the following simple fact about binary trees.

LEMMA 2. *Let a binary tree have $n \geq 1$ branching nodes and therefore $n+1$ leaves. Number the branching nodes with 1 to n recursively in the order: left son–root–right son, and number the leaves with 0 to n from left to right, compare Fig. 1. Then, the path from the root*

- *to leaf 0 contains node 1 and makes a left turn there,*
- *to leaf k, for some $k \in \{1, \ldots, n-1\}$, contains node k and makes a right turn there, and contains node $k+1$ and makes a left turn there,*
- *to leaf n contains node n and makes a right turn there.*

We omit the elementary proof.

We describe the algorithm for a given, fixed ε with $0 < \varepsilon < \frac{1}{2}$. We define

$$n := \lceil 1/(2\varepsilon) \rceil - 1.$$

Then $n \geq 1$. We choose a balanced binary tree with n branching nodes. Remember that "balanced" means that the tree has the shortest possible depth for the number of branching nodes: any path contains at most $\lceil \log_2 (n+1) \rceil$ branching nodes. Now, we number the nodes with 1 to n recursively in the order: left son–root–right son, and we number the leaves with 0 to n from left to right, cf. Fig. 1. We will use this tree for the computation tree. Into branching node k, for $k = 1, \ldots, n$, we place the test "$f(0) f(k/(n+1)) > 0$". In the case of a positive answer the algorithm continues with the right son of the node, in the case of a negative answer with the left son of the node. In leaf k, for $k = 0, \ldots, n$, the algorithm gives as the result the value

$$\text{out}_k := \frac{2k+1}{2n+2}.$$

This ends the description of the algorithm.

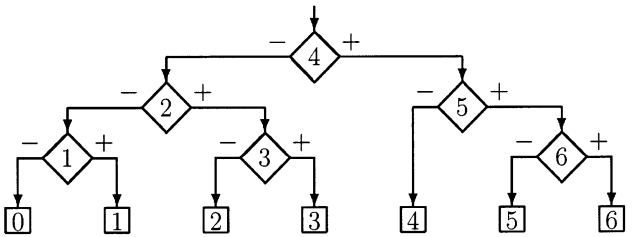

FIG. 1. A balanced binary tree with 6 binary nodes, numbered recursively in the order: left son–root–right son.

We show that it satisfies all the requirements. First of all, the computation tree contains exactly $n = \lceil 1/(2\varepsilon) \rceil - 1$ comparison nodes altogether. Since it is balanced, each path contains at most $\lceil \log_2(n+1) \rceil$ comparison nodes. Let us apply the algorithm to some function $f \in F$. After finitely many steps we arrive in some leaf k, for some $k \in \{0, \ldots, n\}$, and obtain the result $\text{out}_k = (2k+1)/(2n+2)$. We claim that this is a $1/(2n+2)$-approximation of a zero of f. Because of $1/(2n+2) \leqslant \varepsilon$ it is also an ε-approximation of a zero of f. We distinguish three cases.

Case I: *We reach leaf* $k = 0$. Then, according to Lemma 2, during the computation we must have performed the test "$f(0)f(1/n+1)) > 0$" and obtained a negative answer. Thus, $f(0)f(1/(n+1)) \leqslant 0$. This implies that the interval $[0, 1/(n+1)]$ contains a zero of f. Hence, the number $\text{out}_0 = 1/(2n+2)$ is a $1/(2n+2)$-approximation of a zero of f.

Case II: *We reach a leaf k for some* $k \in \{1, \ldots, n-1\}$. Then, according to Lemma 2, we know $f(0)f(k/(n+1)) > 0$ and $f(0)f((k+1)/(n+1)) \leqslant 0$. This implies that the interval $[k/(n+1), (k+1)/(n+1)]$ contains a zero of f. Hence, the number $\text{out}_k = (2k+1)/(2n+2)$ is a $1/(2n+2)$-approximation of a zero of f.

Case III: *We reach leaf* $k = n$. Then, according to Lemma 2, we know $f(0)f(n/(n+1)) > 0$. Since by assumption $f(0)f(1) < 0$, we conclude that $f(n/(n+1))f(1) < 0$. This implies that the interval $[n/(n+1), 1]$ contains a zero of f. Hence, the number $\text{out}_n = (2n+1)/(2n+2)$ is a $1/(2n+2)$-approximation of a zero of f.

This ends the proof of the correctness of the algorithm.

3.2. The Algorithm for Increasing Functions

In this section, we prove the upper bound

$$\text{comp}_{\text{total}}(F_{\text{inc}}, \varepsilon) \leqslant 1$$

for arbitrary ε with $0 < \varepsilon < \frac{1}{2}$. That is, we show that for arbitrary $0 < \varepsilon < \frac{1}{2}$ there is an algebraic oracle computation tree that has only one test and computes an ε-approximation of the zero of f for any $f \in F_{\text{inc}}$. Of course, this implies also $\text{comp}_{\text{path}}(F_{\text{inc}}, \varepsilon) \leqslant 1$.

We shall use two polynomials which act almost like characteristic functions on certain intervals. For arbitrary δ with $0 < \delta < 1$ we fix two polynomials p_δ and q_δ in one variable with the following properties.

(1) p_δ satisfies

$$|p_\delta(x)| \leqslant 1 \qquad \text{for } x \in [-2, 3],$$

$$1 - \delta \leqslant p_\delta(x) \leqslant 1 \qquad \text{for } x \in [-2, -1/3],$$

$$|p_\delta(x)| \leqslant \delta \qquad \text{for } x \in [1/2, 3].$$

(2) q_δ satisfies

$$|q_\delta(x)| \leqslant 1 \qquad \text{for } x \in [-3, 2],$$

$$|q_\delta(x)| \leqslant \delta \qquad \text{for } x \in [-3, -1/2],$$

$$1 - \delta \leqslant q_\delta(x) \leqslant 1 \qquad \text{for } x \in [1/3, 2].$$

Note that such polynomials p_δ and q_δ exist for arbitrary δ with $0 < \delta < 1$ since, due to the Weierstrass approximation theorem, the univariate polynomials form a dense subset of $C(I)$ for any compact interval I. For example, in order to obtain a suitable function p_δ, it is sufficient to take a polynomial that approximates with precision $\delta/2$ the continuous, piecewise linear function from $[-2, 3]$ to $\mathbb{R}$ with constant value $1 - \delta/2$ on $[-2, -\frac{1}{3}]$, with value $(1 - \delta/2)\frac{6}{5}(\frac{1}{2} - x)$ on $[-\frac{1}{3}, \frac{1}{2}]$ and with constant value 0 on $[\frac{1}{2}, 3]$. In addition to p_δ and q_δ we shall use the two fractional transformations r and s defined in the following two lemmas. They map certain pairs of values into intervals on which p_δ or q_δ act almost like characteristic functions. They are illustrated in Fig. 2. We omit the proofs of the lemmas because they are entirely elementary.

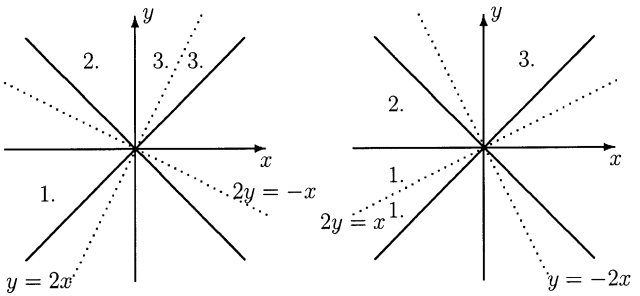

FIG. 2. About the fractional transformations r and s in Lemmas 3 and 4.

LEMMA 3. *For two real numbers x and y with $x \leqslant y$ and*

$$r(x, y) := \frac{-2x + y}{x + 2y},$$

the following three statements are true:

1. *if* $x < 0$ *and* $y \leqslant 0$, *then* $-2 \leqslant r(x, y) \leqslant -\frac{1}{3}$,
2. *if* $x \leqslant 0 \leqslant x + y$ *and* $0 < y$, *then* $\frac{1}{2} \leqslant r(x, y) \leqslant 3$,
3. *if* $0 \leqslant x$ *and* $0 < y$, *then* $-\frac{1}{3} \leqslant r(x, y) \leqslant \frac{1}{2}$.

LEMMA 4. *For two real numbers x and y with $x \leqslant y$ and*

$$s(x, y) := \frac{-x + 2y}{2x + y},$$

the following three statements are true:

1. *if* $x < 0$ *and* $y \leqslant 0$, *then* $-\frac{1}{2} \leqslant s(x, y) \leqslant \frac{1}{3}$,
2. *if* $x < 0$ *and* $x + y \leqslant 0 \leqslant y$, *then* $-3 \leqslant s(x, y) \leqslant -\frac{1}{2}$,
3. *if* $0 \leqslant x$ *and* $0 < y$, *then* $\frac{1}{3} \leqslant s(x, y) \leqslant 2$.

Now we describe the algorithm for a given fixed ε with $0 < \varepsilon < \frac{1}{2}$. We fix a positive integer n with $\varepsilon > 1/(2n + 2)$ and choose a $\delta \in (0, 1)$ small enough such that

$$1 - (1 - \delta)^n \leqslant \varepsilon - 1/(2n + 2).$$

For any function $f \in F_{\text{inc}}$ the algorithm computes

$$x_i := f\left(\frac{i}{n + 1}\right) \qquad \text{for } i = 1, \ldots, n + 1$$

and

$$y_i := x_i(x_i + x_{i+1}) \qquad \text{for } i = 1, \ldots, n.$$

If

$$\prod_{i=1}^{n} y_i \leqslant 0,$$

then it gives as the result

$$\text{out}_{\leqslant} := \frac{3}{2n + 2} + \frac{1}{n + 1} \sum_{i=1}^{n-1} \prod_{j=1}^{i} p_\delta(r(x_j, x_{j+1})).$$

Otherwise it gives as the result

$$\text{out}_{>} := \frac{2n+1}{2n+2} - \frac{1}{n+1} \sum_{i=1}^{n} \prod_{j=n-i+1}^{n} q_\delta(s(x_j, x_{j+1})).$$

This ends the description of the algorithm.

Before we prove the correctness of the algorithm we give some explanations. An important point in the algorithm is that at most one of the factors y_i in the product $\prod_{j=1}^{n} y_i$ can be less than or equal to zero. This is due to the fact that f is increasing, hence $x_i < (x_i + x_{i+1})/2 < x_{i+1}$ for $i \in \{1, \ldots, n\}$. If $\prod_{i=1}^{n} y_i \leqslant 0$, then there is a number $i_0 \in \{1, \ldots, n\}$ with $y_{i_0} < 0$. Then $x_{i_0} \leqslant 0 \leqslant (x_{i_0} + x_{i_0+1})/2$, and the zero of f lies in the interval $[i_0/(n+1), (i_0+1)/(n+1)]$. The idea behind the formula for $\text{out}_{\leqslant}$ is that in the case $y_{i_0} \leqslant 0$ the term $p_\delta(r(x_j, x_{j+1}))$ will be close to 1 for $j < i_0$, close to 0 for $j = i_0$, and have absolute value at most 1 for $j > i_0$. Thus, the product $\prod_{j=1}^{i} p_\delta(r(x_j, x_{j+1}))$ will be close to 1 for $i < i_0$, and close to 0 for $i \geqslant i_0$. If, on the other hand, $\prod_{i=1}^{n} y_i > 0$, then $y_i > 0$ for all $i \in \{1, \ldots, n\}$, hence, either $0 < x_1$, or there must be a number $l_0 \in \{1, \ldots, n\}$ with $(x_{l_0} + x_{l_0+1})/2 < 0 < x_{l_0+1}$. The idea behind the formula for $\text{out}_{>}$ is similar to the idea behind the formula for $\text{out}_{\leqslant}$.

We come to the correctness proof of the algorithm. Let $f \in F_{\text{inc}}$. Then f has a unique zero. We distinguish three cases.

Case 1: $\prod_{i=1}^{n} y_i \leqslant 0$. Then there is an index $i_0 \in \{1, \ldots, n\}$ with $y_{i_0} \leqslant 0$. Since f is increasing, this implies

$$x_{i_0} \leqslant 0 \leqslant (x_{i_0} + x_{i_0+1})/2 < x_{i_0+1}.$$

Hence, the number $(2_{i_0} + 1)/(2n+2)$ is a $1/(2n+2)$-approximation of the zero of f. Furthermore, we obtain

$$x_j < x_{j+1} \leqslant 0 \qquad \text{for all } j < i_0.$$

$$x_j > 0 \qquad \text{for all } j > i_0.$$

Lemma 3 gives us

$$1 - \delta \leqslant p_\delta(r(x_j, x_{j+1})) \leqslant 1 \qquad \text{for } j < i_0,$$

$$|p_\delta(r(x_{i_0}, x_{i_0+1}))| \leqslant \delta,$$

$$|p_\delta(r(x_j, x_{j+1}))| \leqslant 1 \qquad \text{for all } j.$$

We conclude

$$\begin{aligned}
&\left|\mathrm{out}_{\leqslant} - \frac{2i_0+1}{2n+2}\right| \\
&= \frac{1}{n+1}\left|\sum_{i=1}^{n-1}\prod_{j=1}^{i} p_\delta(r(x_j,x_{j+1})) - (i_0-1)\right| \\
&\leqslant \frac{1}{n+1}\left(\sum_{i=1}^{i_0-1}\left|\prod_{j=1}^{i} p_\delta(r(x_j,x_{j+1})) - 1\right| + \sum_{i=i_0}^{n-1}\left|\prod_{j=1}^{i} p_\delta(r(x_j,x_{j+1}))\right|\right) \\
&\leqslant \frac{1}{n+1}\left(\sum_{i=1}^{i_0-1}|(1-\delta)^i - 1| + \sum_{i=i_0}^{n-1}\delta\right) \\
&\leqslant \frac{1}{n+1}\left(\sum_{i=1}^{i_0-1}(1-(1-\delta)^{n-1}) + \sum_{i=i_0}^{n-1}(1-(1-\delta)^{n-1})\right) \\
&= \frac{n-1}{n+1}(1-(1-\delta)^{n-1}) \\
&\leqslant \varepsilon - \frac{1}{2n+2}.
\end{aligned}$$

Hence, $\mathrm{out}_{\leqslant}$ is an ε-approximation of the zero of f.

Case II: $\prod_{i=1}^{n} y_i > 0$ *and* $x_1 < 0$. Then $x_i \neq 0$ for all i. We define

$$l_0 := \min\{i \in \{1,\ldots,n\} \mid x_{i+1} > 0\}.$$

Then,

$$\begin{aligned}
x_j &< 0 \qquad \text{for all } j \leqslant l_0, \\
x_j &> 0 \qquad \text{for all } j > l_0.
\end{aligned}$$

Hence, the number $(2l_0+1)/(2n+2)$ is a $1/(2n+2)$-approximation of the zero of f. Due to $x_j > 0$ for all $j > l_0$ we have $y_j > 0$ for all $j > l_0$. Due to $x_j < 0$ for all $j \leqslant l_0$ we have also $y_j > 0$ for all $j < l_0$. But since $\prod_{i=1}^{n} y_i > 0$, we conclude that also the remaining factor in this product must be greater than 0: $y_{l_0} > 0$. Together with $x_{l_0} < 0$ this implies

$$x_{l_0} + x_{l_0+1} < 0 < x_{l_0+1}.$$

From Lemma 4 we obtain

$$\begin{aligned}
|q_\delta(s(x_{l_0}, x_{l_0+1}))| &\leqslant \delta, \\
1-\delta \leqslant q_\delta(s(x_j,x_{j+1})) &\leqslant 1 \qquad \text{for } j > l_0, \\
|q_\delta(s(x_j,x_{j+1}))| &\leqslant 1 \qquad \text{for all } j.
\end{aligned}$$

We conclude

$$\begin{aligned}
&\left|\text{out}_> - \frac{2l_0+1}{2n+2}\right| \\
&\quad = \frac{1}{n+1}\left|n - l_0 - \sum_{i=1}^{n} \prod_{j=n-i+1}^{n} q_\delta(s(x_j, x_{j+1}))\right| \\
&\quad \leqslant \frac{1}{n+1}\left(\sum_{i=1}^{n-l_0} \left|1 - \prod_{j=n-i+1}^{n} q_\delta(s(x_j, x_{j+1}))\right| \right. \\
&\qquad \left. + \sum_{i=n-l_0+1}^{n} \left| \prod_{j=n-i+1}^{n} q_\delta(s(x_j, x_{j+1}))\right| \right) \\
&\quad \leqslant \frac{1}{n+1}\left(\sum_{i=1}^{n-l_0} |(1-(1-\delta)^i| + \sum_{i=n-l_0+1}^{n} \delta \right) \\
&\quad \leqslant \frac{n}{n+1}(1-(1-\delta)^n) \\
&\quad \leqslant \varepsilon - \frac{1}{2n+2}.
\end{aligned}$$

Hence, $\text{out}_>$ is an ε-approximation of the zero of f.

Case III: $\prod_{i=1}^{n} y_i > 0$ *and* $x_i > 0$. Actually, $x_1 > 0$ implies $\prod_{i=1}^{n} y_i > 0$. The number $1/(2n+2)$ is a $1/(2n+2)$-approximation of the zero of f. Furthermore, we obtain from Lemma 4

$$|q_\delta(s(x_j, x_{j+1})) - 1| \leqslant \delta \qquad \text{for } 1 \leqslant j \leqslant n.$$

As in the last calculations in Case II one obtains

$$\begin{aligned}
\left|\text{out}_> - \frac{1}{2n+2}\right| &\leqslant \frac{1}{n+1} \sum_{i=1}^{n} \left|1 - \prod_{j=n-i+1}^{n} q_\delta(s(x_j, x_{j+1}))\right| \\
&\leqslant \frac{1}{n+1} \sum_{i=1}^{n} |1-(1-\delta)^i| \\
&\leqslant \varepsilon - \frac{1}{2n+2}.
\end{aligned}$$

Hence, the number $\text{out}_<$ is an ε-approximation of the zero of f.

This ends the proof of the correctness of the algorithm.

3.3. The Algorithm for Nondecreasing Functions

In this section, we prove the upper bounds

$$\mathrm{comp}_{\mathrm{total}}(F_{\mathrm{nd}}, \varepsilon) \leqslant \lfloor \log_2(1/\varepsilon) \rfloor$$

and

$$\mathrm{comp}_{\mathrm{path}}(F_{\mathrm{nd}}, \varepsilon) \leqslant \lceil \log_2(\lfloor \log_2(1/\varepsilon) \rfloor + 1) \rceil$$

for arbitrary ε with $0 < \varepsilon < \frac{1}{2}$. We show that for arbitrary $0 < \varepsilon < \frac{1}{2}$ there is an algebraic oracle computation tree that has only $\lfloor \log_2(1/\varepsilon) \rfloor$ branching nodes in total and at most $\lceil \log_2(\lfloor \log_2(1/\varepsilon) \rfloor + 1) \rceil$ branching nodes on any path and computes an ε-approximation of a zero of f for any $f \in F_{\mathrm{nd}}$.

We will use again the polynomials p_δ and q_δ for a suitable δ as well as the fractional transformations r and s which were introduced in Section 3.2. Furthermore, we will use again Lemma 2 in Section 3.1 about binary trees.

We describe the algorithm for a given fixed ε with $0 < \varepsilon < \frac{1}{2}$. We define

$$n := \lfloor \log_2(1/\varepsilon) \rfloor$$

and choose a number $\delta \in (0, 1)$ small enough such that

$$1 - (1 - \delta)^{2^n - 1} \leqslant \varepsilon - 1/2^{n+1}.$$

As in Section 3.1 we choose a balanced binary tree with n branching nodes. Hence, any path contains at most $\lceil \log_2(n + 1) \rceil$ branching nodes. Now, we number the nodes with 1 to n recursively in the order: left son–root–right son, and we number the leaves with 0 to n from left to right; cf. Fig. 1. We will use this tree as the computation tree. Into branching node k we will place a test of the form "$T_k(f) \leqslant 0$" where $T_k(f)$ will be defined below. In the case of a positive answer the algorithm continues with the right son of the node, in the case of a negative answer with the left son of the node.

For any function $f \in F_{\mathrm{nd}}$ the algorithm computes

$$x_i := f\left(\frac{i}{2^n}\right) \qquad \text{for } i = 1, \ldots, 2^n$$

and

$$y_i := x_i \cdot (x_i + x_{i+1}) \qquad \text{for } i = 1, \ldots, 2^n - 1.$$

For $k = 1, \ldots, n$, the test "$T_k(f) \leqslant 0$" is defined by

$$T_k(f) := \prod_{i=1}^{2^{n+1-k}-1} y_{i \cdot 2^{k-1}}.$$

For $k = 1, \ldots, n$, in leaf k the algorithm gives as the result

$$\mathrm{out}_k(f) := \frac{2^k + 1}{2^{n+1}} + \frac{1}{2^{n-k}} \sum_{i=1}^{2^{n-k}-1} \prod_{j=0}^{i-1} p_\delta(r(x_{2^{k-1}+j\cdot 2^k}, x_{(j+1)\cdot 2^k})),$$

and in leaf 0 it gives as the result

$$\mathrm{out}_0(f) := \frac{2^{n+1} - 1}{2^{n+1}} - \frac{1}{2^n} \sum_{i=1}^{2^n-1} \sum_{j=2^n-i}^{2^n-1} q_\delta(s(x_j, x_{j+1})).$$

This ends the description of the algorithm.

The algorithm is based on the same ideas as the algorithm for increasing functions in Section 3.2. There is an additional difficulty: in the correctness proof for the algorithm for increasing functions we were able to conclude in the case $\prod_{i=1}^{n} y_i \leqslant 0$ that exactly one factor y_i in this product is less than or equal to zero. Here we need something similar. But the input function f does not need to be increasing. We only know that it is nondecreasing. Nevertheless, we obtain the desired statement since, when the computation ends in a node $k \in \{1, \ldots, n-1\}$, we know not only $T_k(f) \leqslant 0$ but also $T_{k+1}(f) > 0$.

We come to the correctness proof for the algorithm. Let $f \in F_{\mathrm{nd}}$. We apply the algorithm of f and reach one of the leaves of the computation tree. We distinguish three cases.

Case I: *We reach leaf* 0. Then, according to Lemma 2, the test "$T_1(f) \leqslant 0$" must have had a negative answer, that is, we have

$$\prod_{i=1}^{2^n-1} y_i > 0.$$

This case can be treated in the same way as the union of Cases II and III in the proof of the correctness of the algorithm for increasing functions in Section 3.2. Using the same arguments as before, one shows that $\mathrm{out}_0(f)$ is an ε-approximation of a zero of f.

Case II: *There is some number* $k_0 \in \{1, \ldots, n-1\}$ *such that we reach leaf* k_0. Then, according to Lemma 2, the test "$T_{k_0}(f) \leqslant 0$" must have had a positive answer, and the test "$T_{k_0+1}(f) \leqslant 0$" must have had a negative

answer, that is, we have

$$\prod_{i=1}^{2^{n+1-k_0}-1} y_{i2^{k_0-1}} \leqslant 0 \quad \text{and} \quad \prod_{i=1}^{2^{n-k_0}-1} y_{i2^{k_0}} > 0.$$

We conclude that there is some odd $i' \in \{1, 3, \ldots, 2^{n+1-k_0} - 1\}$ with $y_{i'2^{k_0-1}} \leqslant 0$. Defining $i_0 \in \{0, 1, \ldots, 2^{n-k_0} - 1\}$ by $i' = 2i_0 + 1$ we rewrite this as $y_{2^{k_0-1}+i_0 2^{k_0}} \leqslant 0$. Since f is nondecreasing this means

$$x_{2^{k_0-1}+i_0 2^{k_0}} \leqslant 0 \leqslant \frac{x_{2^{k_0-1}+i_0 2^{k_0}} + x_{2^{k_0-1}+1+i_0 2^{k_0}}}{2} \leqslant x_{2^{k_0-1}+1+i_0 2^{k_0}}.$$

Therefore, the number $(2^{k_0} + i_0 2^{k_0+1} + 1)/2^{n+1}$ is a $1/2^{n+1}$-approximation of a zero of f. Now we show that the output $\mathrm{out}_{k_0}(f)$ in leaf k_0 is an $(\varepsilon - 1/2^{n+1})$-approximation of this number and therefore an ε-approximation of a zero of f. From $x_{2^{k_0-1}+1+i_0 2^{k_0}} \geqslant 0$, from $\prod_{i=1}^{2^{n-k_0}-1} y_{i2^{k_0}} > 0$, and from the fact that f is nondecreasing we conclude that $x_l > 0$ for all $l \geqslant (i_0 + 1)2^{k_0}$, hence, in particular

$$0 < x_{2^{k_0-1}+j2^{k_0}} \leqslant x_{(j+1)2^{k_0}} \qquad \text{for all } j > i_0. \tag{2}$$

Furthermore, we obtain

$$x_{2^{k_0-1}+i_0 2^{k_0}} \leqslant 0 \leqslant x_{2^{k_0-1}+i_0 2^{k_0}} + x_{(i_0+1)2^{k_0}} \quad \text{and} \quad 0 < x_{(i_0+1)2^{k_0}}. \tag{3}$$

In the same way, from $x_{2^{k_0-1}+i_0 2^{k_0}} \leqslant 0$, from $\prod_{i=1}^{2^{n-k_0}-1} y_{i2^{k_0}} > 0$, and from the fact that f is nondecreasing, we conclude that $x_l < 0$ for all $l \leqslant i_0 2^{k_0}$, hence, in particular

$$x_{2^{k_0-1}+j2^{k_0}} \leqslant x_{(j+1)2^{k_0}} < 0 \quad \text{for all } j < i_0. \tag{4}$$

Estimations (2)–(4), and Lemma 3 imply

$$1 - \delta \leqslant p_\delta(r(x_{2^{k_0-1}+j2^{k_0}}, x_{(j+1)2^{k_0}})) \leqslant 1 \qquad \text{for } j < i_0,$$

$$|p_\delta(r(x_{2^{k_0-1}+i_0 2^{k_0}}, x_{(i_0+1)2^{k_0}}))| \leqslant \delta,$$

$$|p_\delta(r(x_{2^{k_0-1}+j2^{k_0}}, x_{(j+1)2^{k_0}}))| \leqslant 1 \quad \text{for all } j.$$

We conclude the proof using the same arguments as in the end of the treatment of Case I in the correctness proof in Section 3.2:

$$\begin{aligned}
&\left|\mathrm{out}_{k_0}(f) - \frac{2^{k_0} + 1 + i_0 2^{k_0+1}}{2^{n+1}}\right| \\
&\quad = \frac{1}{2^{n-k_0}} \left| \sum_{i=1}^{2^{n-k_0}-1} \prod_{j=0}^{i-1} p_\delta(r(x_{2^{k_0-1}+j2^{k_0}}, x_{(j+1)2^{k_0}})) - i_0 \right| \\
&\quad \leqslant \frac{1}{2^{n-k_0}} \left(\sum_{i=1}^{i_0} \left| \prod_{j=0}^{i-1} p_\delta(r(x_{2^{k_0-1}+j2^{k_0}}, x_{(j+1)2^{k_0}})) - 1 \right| \right. \\
&\qquad \left. + \sum_{i=i_0+1}^{2^{n-k_0}-1} \left| \prod_{j=0}^{i=1} p_\delta(r(x_{2^{k_0-1}+j2^{k_0}}, x_{(j+1)2^{k_0}})) \right| \right) \\
&\quad \leqslant \frac{1}{2^{n-k_0}} \left(\sum_{i=1}^{i_0} |(1-\delta)^i - 1| + \sum_{i=i_0+1}^{2^{n-k_0}-1} \delta \right) \\
&\quad \leqslant \varepsilon - \frac{1}{2^{n+1}}.
\end{aligned}$$

Hence, $\mathrm{out}_{k_0}(f)$ is an ε-approximation of a zero of f.

Case III: *We reach leaf n*. Then, according to Lemma 2, the test "$T_n(f) \leqslant 0$" must have had a positive answer, that is, we have

$$y_{2^{n-1}} \leqslant 0.$$

This implies $x_{2^{n-1}} \leqslant 0 \leqslant x_{2^{n-1}+1}$. Hence, the number $(2^n + 1)/2^{n+1}$ is a $1/2^{n+1}$-approximation of a zero of f. The sum in the expression for $\mathrm{out}_n(f)$ is empty, and one obtains

$$\mathrm{out}_n(f) = \frac{2^n + 1}{2^{n+1}}.$$

Thus, $\mathrm{out}_n(f)$ is a $1/2^{n+1}$-approximation of a zero of f and therefore also an ε-approximation of a zero of f.

This ends the correctness proof for the algorithm.

4. THE LOWER BOUNDS

In this section, we prove the lower bounds for the complexities which were stated in Theorem 1.

First of all, we note that for each class G of functions the stated lower bound for $\mathrm{comp}_{\mathrm{path}}(G, \varepsilon)$ follows immediately from the stated lower bound

for $\mathrm{comp}_{\mathrm{total}}(G,\varepsilon)$ since any binary tree with at least n branching nodes in total has at least one path with at least $\lceil \log_2(n+1) \rceil$ branching nodes. Hence, we need to show only the lower bounds for the total number of nodes.

It is sufficient to prove the lower bounds for the classes $G^{-1,1}$, for $G \in \{F, F_{\mathrm{nd}}, F_{\mathrm{inc}}\}$.

In the beginning of Section 3 we have seen that $\mathrm{comp}_{\mathrm{total}}(G,\varepsilon) = 0$ for $\varepsilon \geqslant \frac{1}{2}$, and this matches the lower bounds given in Theorem 1 for $\varepsilon \geqslant \frac{1}{2}$.

Finally, the lower bound

$$\lceil \log_2(\varepsilon^{-1}+2) \rceil - 2 \leqslant \mathrm{comp}_{\mathrm{total}}(F_{\mathrm{nd}},\varepsilon)$$

was already shown by the author [3]. The proof given there applies also to the smaller class $F_{\mathrm{nd}}^{-1,1}$.

Hence, it remains to prove

$$1 \leqslant \mathrm{comp}_{\mathrm{total}}(F_{\mathrm{inc}}^{-1,1},\varepsilon) \quad \text{and} \quad \lceil 1/(2\varepsilon) \rceil - 1 \leqslant \mathrm{comp}_{\mathrm{total}}(F^{-1,1},\varepsilon)$$

for $0 < \varepsilon < \frac{1}{2}$. This will be done in Sections 4.2 and 4.3. In Section 4.1 we formulate a lemma which is useful for both proofs.

4.1. Finding Extended Zeros of Jump Functions

We show that any algebraic oracle computation tree that finds an ε-approximation of a zero of any function in $F_{\mathrm{inc}}^{-1,1}$ or in $F^{-1,1}$ also finds an ε-approximation of an "extended zero" of certain discontinuous functions that can be approximated by functions out of the given class.

First, we introduce some terminology. We call a function $f : [0,1] \to \mathbb{R}$ a *jump function*, if the set

$$D_f := \{x \in [0,1] \mid f \text{ is discontinuous in } x\}$$

of points at which f is not continuous is finite. A number $x_0 \in [0,1]$ is called an *extended zero* of a jump function $f : [0,1] \to \mathbb{R}$, if every neighborhood of x_0 contains a point $x_l \in [0,1]$ with $f(x_l) \leqslant 0$ and a point $x_g \in [0,1]$ with $f(x_g) \geqslant 0$. Such points x_l and x_g do not have to be distinct and may be equal to x_0.

Obviously, any continuous function $f : [0,1] \to \mathbb{R}$ is a jump function. The extended zeros of a continuous function $f : [0,1] \to \mathbb{R}$ are exactly its zeros in the usual sense.

For any $G \in \{F, F^{-1,1}, F_{\mathrm{nd}}, F_{\mathrm{nd}}^{-1,1}, F_{\mathrm{inc}}, F_{\mathrm{inc}}^{-1,1}\}$, one can define the corresponding class JG by considering jump functions instead of continuous functions and keeping the other conditions in the definition of the function class G. We will use the following lemma only for $G \in \{F_{\mathrm{inc}}^{-1,1}, F^{-1,1}\}$.

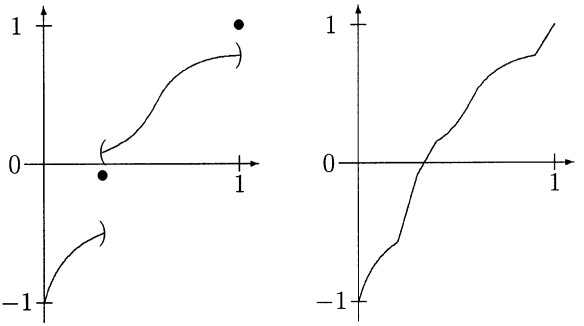

FIG. 3. An increasing jump function f and the function $f_{1/10}$ defined in the proof of Lemma 5.

LEMMA 5. *Fix a class* $G \in \{F, F^{-1,1}, F_{\mathrm{nd}}, F_{\mathrm{nd}}^{-1,1}, F_{\mathrm{inc}}, F_{\mathrm{inc}}^{-1,1}\}$ *and an* $\varepsilon > 0$. *Let* A *be an algebraic oracle computation tree that computes an* ε*-approximation of a zero of* f *for any function* $f \in G$. *Then* A *also computes an* ε*-approximation of an extended zero of* f *for any* $f \in JG$.

Proof. Let f be some function in $JG \setminus G$. For every δ with

$$0 < \delta \leqslant d_f := \min\{|x - y| \mid x, y \in D_f \cup \{0,1\},\ x \neq y\}/2,$$

we define a function $f_\delta : [0,1] \to \mathbb{R}$ as follows; see Fig. 3:

$$f_\delta(x) := \begin{cases} f(\tilde{x}) + (x - \tilde{x})(f(\tilde{x} + \delta) - f(\tilde{x}))/\delta & \text{if there is some } \tilde{x} \in D_f \\ & \quad \text{with } x \in (\tilde{x}, \tilde{x} + \delta), \\ f(\tilde{x}) - (\tilde{x} - x)(f(\tilde{x}) - f(\tilde{x} - \delta))/\delta & \text{if there is some } \tilde{x} \in D_f \\ & \quad \text{with } x \in (\tilde{x} - \delta, \tilde{x}), \\ f(x) & \text{for all other } x. \end{cases}$$

These functions f_δ are continuous. Furthermore, if f is nondecreasing (resp., increasing) then also f_δ is nondecreasing (resp., increasing). And we have $f_\delta(-1) = f(-1)$ and $f_\delta(1) = f(1)$. Hence, $f_\delta \in G$. For δ tending to 0 the functions f_δ converge pointwise to f. Even more: using

$$\delta^{(x)} := \begin{cases} d_f & \text{if } x \in D_f \cup \{0,1\}, \\ \min\{d_f, \mathrm{dist}(x, D_f \cup \{0,1\})\} & \text{if } x \in [0,1] \setminus (D_f \cup \{0,1\}), \end{cases}$$

one sees that $f_\delta(x) = f(x)$ for all $x \in [0,1]$ and all positive $\delta \leqslant \delta^{(x)}$.

We consider the computation which is performed when we apply the algorithm A to f_δ, and we let δ tend to 0. We claim that *there is some positive* δ_f *such that for all positive* $\delta \leqslant \delta_f$ *exactly the same computation is performed*

when A is applied to f_δ. This implies that for $\delta \leqslant \delta_f$, the functions f_δ are all evaluated at the same points $z_1, \ldots, z_l$, and that the values $f_\delta(z_1), \ldots, f_\delta(z_l)$ are independent of δ. Note that this implies $f_\delta(z_i) = f(z_i)$ for $i = 1, \ldots, l$.

We prove this claim by induction over the computation steps. Let us consider some computation step $t+1$ and assume that there is some δ_t such that the computation performed up to step t is identical for all positive $\delta \leqslant \delta_t$. This implies that all points at which f_δ has been evaluated so far and the values of f_δ at these points are the same for all $\delta \leqslant \delta_t$. If at step $t+1$ a numerical computation or a comparison is performed, then it is based on previously obtained values of f_δ and previously performed computations. Hence, it is the same for all positive $\delta \leqslant \delta_t$, and we can set $\delta_{t+1} := \delta_t$. If at step $t+1$ the value of f_δ at some point z is asked for, then, since z must have been computed earlier and is therefore the same for all $\delta \leqslant \delta_t$, we can take $\delta_{t+1} := \min\{\delta^{(z)}, \delta_t\}$. This ends the induction step. Finally, since the computation tree A is finite, there is an upper bound, independent from δ, for the length of computations performed when applying A to f_δ. This implies that the minimum $\delta_f := \min\{\delta_t \mid t \geqslant 1\}$ is positive. Hence, we have proved our assertion: there is some positive δ_f such that for all positive $\delta \leqslant \delta_f$ exactly the same computation is performed when A is applied to f_δ.

If we apply A to f, then, due to the claim just proved, the same computation is performed as for any f_δ with $0 < \delta < \delta_f$. Hence, the computation ends after finitely many steps, and the result—let us call it $\mathrm{out}_A(f)$—is an ε-approximation of a zero of f_δ, for any δ with $0 < \delta \leqslant \delta_f$. For each such δ, let z_δ be a zero of f_δ with $|\mathrm{out}_A(f) - z_\delta| \leqslant \varepsilon$. The set $\{z_{1/n} \mid n \in \mathbb{Z}, n > 0, 1/n \leqslant \delta_f\}$ has an accumulation point z_f in the interval $[0, 1]$. From the definition of f_δ it is clear that z_f is an extended zero of f. Since it is also clear that $|\mathrm{out}_A(f) - z_f| \leqslant \varepsilon$, the lemma is proved. ∎

4.2. *The Lower Bound for Increasing Functions*

In this section, we prove the lower bound

$$1 \leqslant \mathrm{comp}_{\mathrm{total}}(F_{\mathrm{inc}}^{-1,1}, \varepsilon)$$

for $0 < \varepsilon < \frac{1}{2}$.

Let us fix a number ε with $0 < \varepsilon < \frac{1}{2}$. For the sake of a contradiction, we assume that there is an algorithm which does not use any comparison and computes an ε-approximation of the zero of f for any $f \in F_{\mathrm{inc}}^{-1,1}$.

We define a class of jump functions $f_y : [0, 1] \to \mathbb{R}$ for $y \in [-(\sqrt{5} - 1)/2, 0) \cup (0, (\sqrt{5} - 1)/2]$ by

$$f_y(x) := \begin{cases} -1 & \text{if } x = 0, \\ y + y^2(2x - 1) & \text{if } 0 < x < 1, \\ 1 & \text{if } x = 1. \end{cases}$$

These functions f_y are jump functions in $JF_{\text{inc}}^{-1,1}$, cf. Section 4.1. By Lemma 5 the algorithm A will compute an ε-approximation of an extended zero of f_y, when applied to f_y.

When applied to f_y, the algorithm will compute step by step a sequence of real values $v_1, \ldots, v_m$ (depending on y). Each of them can either be a constant, or be given by a rational expression in previously computed values, or be the value of f_y at a previously computed value. One of these values, without loss of generality the last one, v_m, will be the output of the computation. We denote it by $\text{out}_A(f_y)$.

We claim that *for each* $i = 1, \ldots, m$ *there is a rational expression* $r_i(y)$ *in the variable* y *such that* $v_i = r_i(y)$ *for all but finitely many parameters* y.

This claim is proved by induction over $i = 1, \ldots, m$. Let us assume that it is true for all $i<k$ for some k. We distinguish three cases for the value v_k.

Case I. If v_k is a constant, say, c, the assertion is true also for k by setting $r_k(y) := c$.

Case II. Assume that v_k is given by a rational expression in previously computed values, i.e., values of the form v_j with $j<k$. Then we can plug into this expression the rational expressions $r_j(y)$ which we know for the previously computed values by induction hypothesis. In this way, we obtain a rational expression in y. We call it $r_k(y)$. The expressions $r_j(y)$ for the previously computed values v_j gave the correct values v_j except for finitely many y. Hence, we also have $r_k(y) = v_k$ except for finitely many y.

Case III. The last case is the case that v_k is the value of the function f_y at some previously computed value v_j, for some $j<k$, that is, $v_k = f_y(v_j)$. We distinguish three subcases.

Subcase III.I. The rational function $r_j(y)$ is constant with value 0 on its domain of definition. Then we set $r_k(y) := -1$. For all but finitely many y we have $v_j = r_j(y)$, and, hence, $v_k = f_y(v_j) = f_y(r_j(y)) = f_y(0) = -1 = r_k(y)$.

Subcase III.II. The rational function $r_j(y)$ is constant with value 1 on its domain of definition. Then we set $r_k(y) := 1$.

Subcase III.III. The rational function $r_j(y)$ is not constant on its domain of definition or it is constant on its domain of definition but its value is different from 0 and 1. Then, since a nonconstant rational function in one variable can take a certain value only finitely many times, for all but finitely many y we have $v_j = r_j(y)$ and $r_j(y) \notin \{0, 1\}$, and therefore

$$v_k = f_y(v_j) = y + y^2(2r_j(y) - 1).$$

Thus, the assertion is proved by setting $r_k(y) := y + y^2(2r_j(y) - 1)$.

We have proved the claim by induction. Applying it to the output $v_m = \text{out}_A(f_y)$, we obtain a rational expression $r_m(y)$. This rational expression defines a rational function in y which, for all but finitely many y, is equal to the output $v_m = \text{out}_A(f_y)$ of the algorithm, and therefore an ε-approximation

of an extended zero of f_y. The functions f_y have exactly one extended zero: it is given by $z(y) = 1$ for $y < 0$ and by $z(y) = 0$ for $y > 0$. Thus, we have

$$|r_m(y) - z(y)| \leqslant \varepsilon$$

for all but finitely many $y \in [-(\sqrt{5}-1)/2, 0) \cup (0, (\sqrt{5}-1)/2]$. Thus, 0 cannot be a pole of the rational function $r_m(y)$. But $r_m(y)$ can also not be extended continuously into 0 because $\varepsilon < \frac{1}{2}$. Contradiction! Hence, there cannot be an algorithm that computes without comparisons an ε-approximation of the zero of f, for any $f \in F_{\text{inc}}^{-1,1}$.

4.3. The Lower Bound for Arbitrary Functions

In this section, we prove the lower bound

$$\lceil 1/(2\varepsilon) \rceil - 1 \leqslant \text{comp}_{\text{total}}(F^{-1,1}, \varepsilon)$$

for $0 < \varepsilon < \frac{1}{2}$.

First, we prove a lemma related to the Newton polyhedron of a polynomial or a formal power series. It is based on an idea used by Vassiliev [8] and by Grigoriev *et al.* [2]. Let us fix an integer $n \geqslant 1$ and a vector $\bar{a} = (a_1, \ldots, a_n)$ of positive real numbers which are linearly independent over $\mathbb{Q}$. That means that

$$\sum_{i=1}^{n} z_i a_i \neq 0.$$

for any integer vector $\bar{z} = (z_1, \ldots, z_n) \in \mathbb{Z}^n \setminus \{(0, \ldots, 0)\}$.

LEMMA 6. *Let $r(x_1, \ldots, x_n)$ be a nonzero (i.e., not identically equal to zero) rational function in n variables.*

(1) *There exists a number $b > 0$ such that for every sign vector $(\sigma_1, \ldots, \sigma_n) \in \{-1, 1\}^n$, the numbers $r(\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})$ for $t \in (0, b]$ are well defined (not plus or minus infinity) and nonzero and have the same sign, i.e., their sign depends only on $(\sigma_1, \ldots, \sigma_n)$ but not on t.*

(2) *Either $\lim_{t \searrow 0} |r(\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})| = \infty$ for all sign vectors $(\sigma_1, \ldots, \sigma_n) \in \{-1, 1\}^n$, or the limit $\lim_{t \searrow 0} r(\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})$ exists (is finite) for all sign vectors $(\sigma_1, \ldots, \sigma_n)$, and it has the same absolute value for all.*

Proof. We can write the rational function $r(x_1, \ldots, x_n)$ as the quotient of two nonzero polynomials in $x_1, \ldots, x_n$:

$$r(x_1, \ldots, x_n) = p(x_1, \ldots, x_n)/q(x_1, \ldots, x_n).$$

Since the numbers $a_1, \ldots, a_n$ are linearly independent over $\mathbb{Q}$, there is a unique monomial $c \cdot x_1^{m_1} \cdot \ldots \cdot x_n^{m_n}$ with nonzero coefficient c in $p(x_1, \ldots, x_n)$ such that

$$\sum_{i=1}^{n} m_i a_i = \min\left\{ \sum_{i=1}^{n} l_i a_i \,|\, \text{the monomial } x_1^{l_1} \cdot \ldots \cdot x_n^{l_n} \cdot \text{ has a nonzero coefficient in } p \right\}$$

(the vector $(m_1, \ldots, m_n)$ is the unique vector in the Newton polyhedron of p in which the lower supporting plane of the Newton polyhedron of p orthogonal to $\bar{a}$ touches the Newton polyhedron). For any sign vector $\bar{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \{-1, 1\}^n$, this monomial dominates all other monomials in p on the curve $(x_1, \ldots, x_n) = (\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})$ for sufficiently small positive t. There is also a uniquely determined monomial $c' \cdot x_1^{m_1'} \cdot \ldots \cdot x_n^{m_n'}$ with nonzero coefficient c' in $q(x_1, \ldots, x_n)$ with the corresponding property. Thus, the behavior of $r(\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})$ for any sign vector $\bar{\sigma}$ and for positive t tending to zero is determined by the term

$$\frac{c}{c'} t^{(m_1 - m_1')a_1 + \cdots + (m_n - m_n')a_n} \prod_{i=1}^{n} \sigma_i^{m_i - m_i'}.$$

Especially, there is a positive number b such that for all sign vectors $\bar{\sigma}$ and all $t \in (0, b]$ the sign of $r(\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})$ is equal to the sign of $c/c' \prod_{i=1}^{n} \sigma_i^{m_i - m_i'}$. The assertions follow. ∎

Let us fix a number ε with $0 < \varepsilon < \frac{1}{2}$. For the sake of a contradiction we assume that there are an integer n with $1 \leqslant n < 1/(2\varepsilon)$ and an algebraic oracle computation tree A with exactly $n - 1$ comparison nodes in total which computes an ε- approximation of a zero of f for any $f \in F^{-1,1}$. We will derive a contradiction in a similar way as in the proof in Section 4.2. Therefore, we will consider jump functions of the following form: for a vector $\bar{s} = (s_0, \ldots, s_n) \in \mathbb{R}^{n+1}$ with

$$0 < s_0 < s_1 < \cdots < s_n \leqslant 1$$

and a vector $\bar{y} = (y_1, \ldots, y_n) \in \mathbb{R}^n$ we define the jump function $f_{\bar{s},\bar{y}} : [0, 1] \to \mathbb{R}$ by

$$f_{\bar{s},\bar{y}}(x) := \begin{cases} -1 & \text{if } 0 \leqslant x < s_0, \\ y_i & \text{if } s_{i-1} \leqslant x < s_i \quad \text{for some } i \in \{1, \ldots, n\}, \\ 1 & \text{if } s_n \leqslant x \leqslant 1. \end{cases}$$

These functions $f_{\bar{s},\bar{y}}$ are obviously elements of $JF^{-1,1}$, cf. Section 4.1. By Lemma 5 the computation tree computes an ε-approximation of an extended zero for any such function $f_{\bar{s},\bar{y}}$.

For the rest of this section we fix a vector $\bar{a} = (a_1, \ldots, a_n)$ of positive real numbers which are linearly independent over $\mathbb{Q}$.

First, we shall show that for a certain class of input functions of the form $f_{\bar{s},\bar{y}}$ the values computed in the algorithm are given by rational expressions in $y_1, \ldots, y_n$ where the expression for a value depends only on the corresponding node in the computation tree in which this value is computed.

LEMMA 7. *Let the* (*unary*) *nodes of the tree A be numbered from* 1 *to*, *say*, *m*, *and denote the value computed in the unary node i by* v_i.

There exist

(1) *a vector* $\bar{s} = (s_0, \ldots, s_n) \in \mathbb{R}^{n+1}$ *with* $0 < s_0$, *with* $s_i + 2\varepsilon < s_{i+1}$ *for all* $i \in \{0, \ldots, n-1\}$, *and with* $s_n \leqslant 1$,

(2) *rational expressions* $r^{[k]}(y_1, \ldots, y_n)$ *in n variables*, *for* $k = 1, \ldots, m$,

(3) *a positive real number* $b(\bar{s})$,

with the following property for all unary nodes in the computation tree, *for all sign vectors* $\bar{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \{-1, 1\}^n$, *and for all* $t \in (0, b(\bar{s})]$: *if the unary node i lies on the computation path on input* $f_{\bar{s},\bar{y}}$ *where*

$$\bar{y} = (y_1, \ldots, y_n) \coloneqq (\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n}),$$

then

$$v_i = r^{[i]}(y_1, \ldots, y_n).$$

Proof. In order to prove the lemma we shall prove a slightly stronger claim by induction for subtrees which are obtained by cutting off leaves or branches. We introduce some notation. Let us say that an algebraic oracle computation tree is *well defined on input* f for some function $f : [0, 1] \to \mathbb{R}$, if its evaluation on input f leads to a computation path ending in a leaf, without divisions by 0, and such that any value v for which $f(v)$ is computed lies in the interval $[0, 1]$. We consider computation trees with the property that they are

$$\textit{well defined on input } f \textit{ for any } f \in JF^{-1,1}. \tag{5}$$

Note that by Lemma 5 all computation trees that compute an ε-approximation of a zero for any function $f \in F^{-1,1}$ have Property (5). Also, for any tree with Property (5), any subtree which is obtained by cutting off leaves or branches has Property (5). Furthermore, we use

$\delta := 1/(2n) - \varepsilon$ and

$$S_\varepsilon := \{\bar{s} = (s_0, \ldots, s_n) \in (0,1)^{n+1} \mid |s_i - i/n| < \delta \quad \text{for all } i \in \{0, \ldots, n\}\}.$$

Let us consider a computation tree B with Property (5). We number the unary nodes in B in some order, say, from 1 to m, and denote the value computed in the unary node with number i by v_i. We shall prove by induction over the size of the computation tree that there exist

(1) a nonempty open subset $S \subseteq S_\varepsilon$,
(2) rational expressions $r^{[k]}(y_1, \ldots, y_n)$ in n variables, for $k = 1, \ldots, m$,
(3) a function $b : S \to \mathbb{R}_+$ (where $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x > 0\}$),

with the following property for all unary nodes in the computation tree, for all $\bar{s} \in S$, for all sign vectors $\bar{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \{-1, 1\}^n$, and for all $t \in (0, b(\bar{s})]$: if the unary node i lies on the computation path on input $f_{\bar{s},\bar{y}}$, where

$$\bar{y} = (y_1, \ldots, y_n) := (\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n}),$$

then

$$v_i = r^{[i]}(y_1, \ldots, y_n). \tag{6}$$

In order to deduce the assertion of the lemma from this claim observe that the computation tree A has property (5). An arbitrary vector $\bar{s} = (s_0, \ldots, s_n) \in S_\varepsilon$ satisfies $0 < s_0$, $s_i + 2\varepsilon < s_{i+1}$ for all $i \in \{0, \ldots, n-1\}$, and $s_n < 1$. Thus, it is clear that the lemma is proved once we have proved this claim. We prove this claim by induction over the size of the computation tree.

Let B' be an algebraic oracle computation tree with Property (5) which has exactly $m+1$ unary nodes. We number them with numbers from 1 to $m+1$ such that node $m+1$ does not lie on a path from the root to any node i with $i \in \{1, \ldots, m\}$. We denote the value computed in node j by v_j, for $j = 1, \ldots, m+1$. Let B be the computation tree obtained by cutting off a branch containing the unary node $m+1$ but no other unary node. Then B has Property (5). Therefore, by induction hypothesis there are a list of rational expressions $r^{[k]}(y_1, \ldots, y_n)$ in n variables, for $k = 1, \ldots, m$, a nonempty open subset $S \subseteq S_\varepsilon$, and a function $b : S \to \mathbb{R}_+$, with Property (6) for the tree B, that is, for the unary nodes $1, \ldots, m$. We shall show that there are a rational expression $r^{[m+1]}(y_1, \ldots, y_n)$ in n variables, a nonempty open subset $S' \subseteq S$, and a function $b' : S \to \mathbb{R}_+$ with $b'(\bar{s}) \leqslant b(\bar{s})$ for all $\bar{s} \in S'$ such that the following objects: the set S', the list of rational expressions $r^{[k]}(y_1, \ldots, y_n)$ in n variables, for $k = 1, \ldots, m+1$, and the function b', have Property (6) for the tree B', that is, for the unary nodes $1, \ldots, m+1$.

For the unary nodes $1,\ldots,m$ these objects have these properties already by induction hypothesis since we choose $S' \subseteq S$ and $b'(\bar{s}) \leqslant b(\bar{s})$ for all $\bar{s} \in S'$.

For the node $m+1$ and the value v_{m+1} we distinguish two cases.

Case I: There are a rational expression r and indices $i_1,\ldots,i_{j(m+1)} \in \{1,\ldots,m\}$ with $v_{m+1} = r(v_{i_1},\ldots,v_{i_{j(m+1)}})$ (this includes the case that v_{m+1} is a constant). Then, we can plug into r the rational expressions $r^{[i_1]},\ldots,r^{[i_{j(m+1)}]}$ in $y_1,\ldots,y_n$ which we have by induction hypothesis for the previously computed values $v_{i_1},\ldots,v_{i_{j(m+1)}}$. In this way, we obtain a rational expression in $y_1,\ldots,y_n$ which we call $r^{[m+1]}$. Let us fix some $\bar{s} \in S$, some sign vector $\bar{\sigma}$, and some $t \in (0, b(\bar{s})]$ such that the unary node $m+1$ lies on the computation path of $f_{\bar{s},\bar{y}}$ for $\bar{y} = (\sigma_1 t^{a1},\ldots,\sigma_n t^{a_n})$. Then, on input $f_{\bar{s},\bar{y}}$ also the values $v_{i_1},\ldots,v_{i_{j(m+1)}}$ are computed and satisfy $v_{i_1} = r^{[i_1]}(\bar{y}),\ldots,v_{i_{j(m+1)}} = r^{[i_{j(m+1)}]}(\bar{y})$ by induction hypothesis. Hence, on input $f_{\bar{s},\bar{y}}$ we also have $v_{m+1} = r^{[m+1]}(\bar{y})$. Thus, the assertion is proved by defining $S' := S$ and $b' := b$.

Case II: The unary node $m+1$ contains a function evaluation at a previously computed value, that is, there is an index $j \in \{1,\ldots,m\}$ such that $v_{m+1} = f(v_j)$ where f is the input function. In this case, we consider the behavior of $r^{[j]}(\sigma_1 t^{a_1},\ldots,\sigma_n t^{a_n})$ for t tending to zero. By the second statement of Lemma 6, we know that the limit $\lim_{t \searrow 0} r^{[j]}(\sigma_1 t^{a_1},\ldots,\sigma_n t^{a_n})$ either does not exist (is not finite) for all sign vectors $\bar{\sigma}$ or it exists for all sign vectors and has the same absolute value for all. Thus, the value

$$z := \lim_{t \searrow 0} |r^{[j]}(\sigma_1 t^{a_1},\ldots,\sigma_n t^{a_n})|.$$

is a well-defined element in $\{x \in \mathbb{R} \mid x \geqslant 0\} \cup \{\infty\}$ and independent of the sign vector $\bar{\sigma} = (\sigma_1,\ldots,\sigma_n)$. We distinguish four subcases depending on the value of z.

Subcase II.I: $z = \infty$ or $z \in \{x \in \mathbb{R} \mid x > 1\}$. Then there is a positive number $\tilde{b}$ such that

$$|r^{[j]}(\sigma_1 t^{a_1},\ldots,\sigma_n t^{a_n})| > 1$$

for all sign vectors $\bar{\sigma}$ and all $t \in (0, \tilde{b}]$. Since the considered input functions are defined only on the interval $[0,1]$, they may be evaluated only there. Hence, the unary node $m+1$ cannot lie on the computation path on input $f_{\bar{s},\bar{y}}$ for any $\bar{s} \in S$ and any $\bar{y} = (\sigma_1 t^{a_1},\ldots,\sigma_n t^{a_n})$ with $\bar{\sigma} \in \{-1,1\}^n$ and $t \in (0, \tilde{b}]$. Thus, the assertion is proved by defining $S' := S$, $b'(\bar{s}) := \min\{\tilde{b}, b(\bar{s})\}$ for all $\bar{s} \in S'$, and $r^{[m+1]}$ arbitrary, for example $r^{[m+1]}(\bar{y}) := 0$.

Subcase II.II: $z \leqslant 1$ and for all $i \in \{0,\ldots,n\}$ we have $|z - i/n| \geqslant \delta$. Then there is a (uniquely defined) index $i' \in \{1,\ldots,n\}$ with $z \in [(i'-1)/n + \delta, i'/n - \delta]$. Hence, for each $\bar{s} \in S$ we can choose a positive number $b'(\bar{s}) \leqslant b$

$(\bar{s})$ such that

$$|r^{[j]}(\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})| \in (s_{i'-1}, s_{i'})$$

for all sign vectors $\bar{\sigma}$ and all $t \in (0, b'(\bar{s})]$. Furthermore, we define $S' := S$, and $r^{[m+1]}(y_1, \ldots, y_n) := y_{i'}$. We show that with these choices, Property (6) is true for the unary node $m+1$. Let us fix some $\bar{s} \in S'$, some sign vector $\bar{\sigma}$, and some $t \in (0, b'(\bar{s})]$, let us set $\bar{y} := (\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})$, let us consider the input function $f_{\bar{s},\bar{y}}$, and let us assume that the unary node $m+1$ lies on the corresponding computation path. Then also the value v_j must be computed on this computation path. Furthermore, since the input function $f_{\bar{s},\bar{y}}$ is defined only on the interval $[0, 1]$, we must have $v_j \in [0, 1]$. Together with the induction hypothesis $v_j = r^{[j]}(\bar{y})$ and $|r^{[j]}(\bar{y})| \in (s_{i'-1}, s_{i'})$ this implies $v_j \in (s_{i'-1}, s_{i'})$. We conclude

$$v_{m+1} = f_{\bar{s},\bar{y}}(v_j) = y_{i'} = r^{[m+1]}(y_1, \ldots, y_n).$$

Thus, Property (6) is true for the unary node $m+1$ in this subcase.

Subcase II.III. $z \leqslant 1$, there is an $i \in \{0, \ldots, n\}$ with $|z - i/n| < \delta$, (note that there can be only one such index i), and the set

$$\{\bar{s} = (s_1, \ldots, s_n) \in S \mid z < s_i\}$$

is nonempty. This set is of course open. Then we define

$$S' := \{\bar{s} = (s_1, \ldots, s_n) \in S \mid z < s_i\}.$$

For each $\bar{s} \in S'$ we can choose a positive number $b'(\bar{s}) \leqslant b(\bar{s})$ such that

$$|r^{[j]}(\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})| \in (i/n - \delta, s_i)$$

for all $t \in (0, b'(\bar{s})]$. Finally, we define

$$r^{[m+1]}(y_1, \ldots, y_n) := \begin{cases} -1 & \text{if } i = 0, \\ y_i & \text{if } i \in \{1, \ldots, n\}. \end{cases}$$

One shows as in Subcase II.II that Property (6) is true for the unary node $m+1$.

Subcase II.IV. $z \leqslant 1$, there is an $i \in \{0, \ldots, n\}$ with $|z - i/n| < \delta$, (note that there can be only one such index i), and the set

$$\{\bar{s} = (s_1, \ldots, s_n) \in S \mid z < s_i\}$$

is empty. Then we define

$$S' := \{\bar{s} = (s_1, \ldots, s_n) \in S \mid z > s_i\}$$

and observe that this set is nonempty and open. For each $\bar{s} \in S'$ we can choose a positive number $b'(\bar{s}) \leqslant b(\bar{s})$ such that

$$|r^{[j]}(\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})| \in (s_i, i/n + \delta)$$

for all $t \in (0, b'(\bar{s})]$. Finally, we define

$$r^{[m+1]}(y_1, \ldots, y_n) := \begin{cases} y_{i+1} & \text{if } i \in \{0, \ldots, n-1\}, \\ 1 & \text{if } i = n. \end{cases}$$

One shows as in Subcase II.II that Property (6) is true for the unary node $m+1$.

This ends the proof of Lemma 7. ∎

Let us fix a vector $\bar{s}$, a positive real number $b(\bar{s})$, and a sequence of rational expressions, one for each unary node in the computation tree A, with the properties formulated in Lemma 7. From now on we will consider only input functions for A of the form $f_{\bar{s},\bar{y}}$ with $\bar{y} = (\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})$ where $\bar{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \{-1, 1\}^n$ and $t \in (0, b(\bar{s})]$. According to Lemma 7, for such input, any numerical value computed by the algorithm A is given by some rational expression in $y_1, \ldots, y_n$ where these expressions depend only on the corresponding computation node in the tree. This applies especially to any value which is compared with zero in some comparison node and to any value which is the output in some leaf. Let $p^{(j)}(y_1, \ldots, y_n)$ for $j = 1, \ldots, n-1$ denote the rational functions in $y_1, \ldots, y_n$ used in the $n-1$ comparison nodes, and let $q^{(j)}(y_1, \ldots, y_n)$ for $j = 1, \ldots, n$ denote the rational functions in $y_1, \ldots, y_n$ given as the result in the n leaves (the order is not important).

According to the first statement in Lemma 6, for each $j \in \{1, \ldots, n-1\}$ there is a number $b^{(j)} > 0$ such that for $t \in (0, b^{(j)}]$ and $\bar{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \{-1, 1\}^n$ the term $p^{(j)}(\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})$ is nonzero and its sign does not depend on t but only on the sign vector $\bar{\sigma}$. We define

$$\hat{b} := \min\{b^{(1)}, \ldots, b^{(n-1)}, b(\bar{s})\}.$$

Thus, for each sign vector $\bar{\sigma}$ there is a leaf in the computation tree such that the computation of A on input $f_{\bar{s},\bar{y}}$ with $\bar{y} = (\sigma_1 t^{a_1}, \ldots, \sigma_n t^{a_n})$ follows the same computation path and ends in that leaf for all $t \in (0, \hat{b}]$. We are especially interested in the $n+1$ sign vectors $\bar{\sigma}^{(j)} = (\sigma_1^{(j)}, \ldots, \sigma_n^{(j)})$ for $j = 0, \ldots, n$ defined by

$$\sigma_i^{(j)} := \begin{cases} -1 & \text{if } i \leqslant j, \\ 1 & \text{if } i > j. \end{cases}$$

Since the computation tree has only n leaves, there are numbers $k, l \in \{0, \ldots, n\}$ with $k \neq l$ and a leaf of the computation tree such that the computation of A on input $f_{\bar{s},\bar{y}}$ ends in that leaf for all $\bar{y} = (\sigma_1^{(j)} t^{a_1}, \ldots, \sigma_n^{(j)} t^{a_n})$ with $j \in \{k, l\}$ and $t \in (0, \hat{b}]$. The result in that leaf, given by, say, the rational function $q^{(i)}(y_1, \ldots, y_n)$, must be an ε-approximation of an extended zero of $f_{\bar{s},\bar{y}}$. Since any function $f_{\bar{s},\bar{y}}$ with $\bar{y} = (\sigma_1^{(j)} t^{a_1}, \ldots, \sigma_n^{(j)} t^{a_n})$ for positive t (also for arbitrarily small t) has exactly one extended zero: the number s_j, we conclude that for all $t \in (0, \hat{b}]$,

(1) the value $q^{(i)}(\sigma_1^{(k)} t^{a_1}, \ldots, \sigma_n^{(k)} t^{a_n})$ must be an ε-approximation of s_k, and

(2) the value $q^{(i)}(\sigma_1^{(l)} t^{a_1}, \ldots, \sigma_n^{(l)} t^{a_n})$ must be an ε-approximation of s_l.

According to the second statement of Lemma 6, for t tending to 0, both values must converge to some real numbers of the same absolute value. But this contradicts $|s_k - s_l| > 2\varepsilon$.

This ends the proof of $\lceil 1/(2\varepsilon) \rceil - 1 \leqslant \mathrm{comp}_{\mathrm{total}}(F^{-1,1}, \varepsilon)$ for $0 < \varepsilon < \frac{1}{2}$.

REFERENCES

1. L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP completeness, recursive functions and universal machines, *Bull. Amer. Soc.* **21** (1989), 1–46.
2. D. Grigoriev, M. Karpinski, and R. Smolensky, Randomization and the computational power of analytic and algebraic decision trees, *Comput. Complexity* **6** (1996/97), 376–388.
3. P. Hertling, Topological complexity with continuous operations, *J. Complexity* **12** (1996), 315–338.
4. E. Novak, The real number model in numerical analysis, *J. Complexity* **11** (1995), 57–73.
5. E. Novak and H. Woźniakowski, Topological complexity of zero-finding, *J. Complexity* **12** (1996), 380–400.
6. S. Smale, On the topology of algorithms, I, *J. Complexity* **3** (1987), 81–89.
7. V. A. Vassiliev, Topological complexity of root-finding algorithms, *in* "The Mathematics of Numerical Analysis" (J. Renegar *et al.*, Eds.), Lecture Notes in Applied Mathematics, Vol. 32, pp. 831–856, American Mathematical Society, Providence, RI, 1996.
8. V. A. Vassiliev, On decision trees for orthants, *Inform. Process. Lett.* **62** (1997), 265–268.